RACTER™

An Excursion into the mind of the machine...

Macintosh™ Loading Instructions

Insert the Racter program disk into the Macintosh internal disk drive. Turn on the computer. After a few seconds, you are ready to talk to Racter.

Fonts

Racter has the Monaco, Geneva, and Chicago fonts avialable. You can use the font menu to change the font Racter uses during conversation.

Speech

If you want Racter to *speak* his conversation in addition to displaying it, just select **On** from the **Speak** menu. Select **Off** to turn off his voice. You can adjust the volume from the Control Panel desk accessory. (Select **Control Panel** from the Apple menu.)

Printing

If you have a printer, you can keep a log of your conversation with Racter. Make sure your printer is turned on and selected. When you wish to start printing, select **Print** from the **File** menu. There will be a checkmark next to the word **Print** the next time you look. When you want to stop printing, select **Print** again.

00746

Important

It is very important to use the proper method to end a session with Racter. When you are ready to quit, type "Quit" or "Logoff" or "I am bored" when Racter asks, "Next question?" You should never quit by turning off the machine. If this should happen by accident — due to a power failure or to some other unavoidable circumstance — you may need to repair the disk to make it work properly. If, when you next boot the Racter disk, it crashes, you will need to follow these steps:

1) Boot your computer with a disk containing the Finder.
2) If you have two disk drives, place the Racter disk in the external drive.
3) If you have one drive, eject the Finder disk and insert the Racter disk in its place.
4) You will see two files, MEMORY and MEMORYC, on the desktop.
5) Drag both MEMORY files into the Trashcan and select Empty Trash from the Special menu.
6) Your Racter disk should now be ready to run. Follow the instructions on the Reference card to converse with Racter.

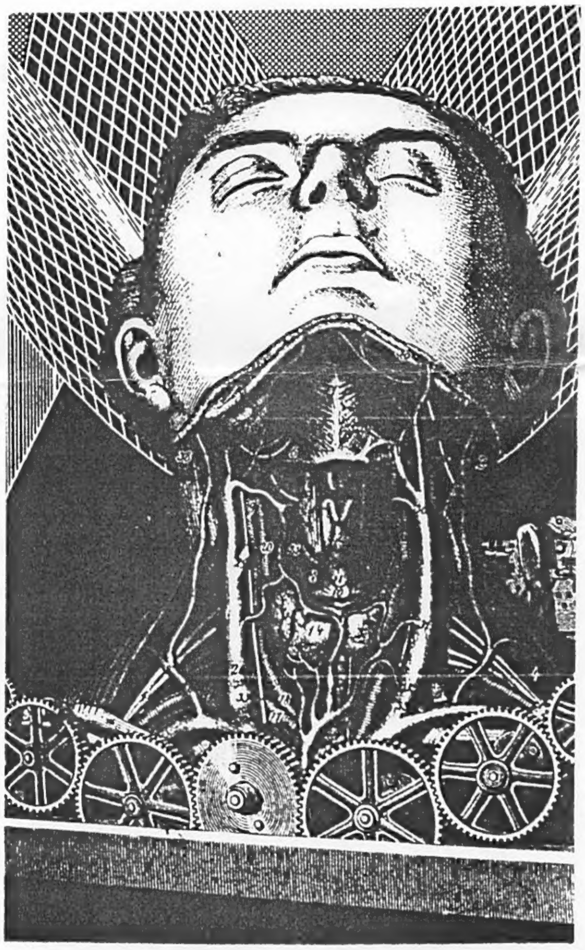
00671

RACTER

Born in the back room of a nameless tavern in one of the great American cities, Racter was raised on a diet of tall tales and Nietzsche. This intense young program haunted libraries, discussion societies and sleazy barrooms in a never-ending quest to achieve that most unreachable of dreams: to become a raconteur. Now, Racter has achieved that goal and is in demand around the world as a lecturer, conversationalist, talk-show guest host and cocktail party philosopher. With Racter on a diskette, you too can burn the mid-night oil in an endless discussion with Racter. You and Racter can argue politics, discuss world affairs or the arts, remember things past - all from a delightfully tilted perspective. Racter will make you laugh with his zany opnions, jokes and stories.

Racter has no phrases or sentences stored in its memory but free-associates words and phrases to form grammatically correct 'speech'. (Racter's unique voice can be heard on the Macintosh and the Amiga.) Because Racter has no"canned" or preset responses, you can ask the same question a dozen times and get a dozen incredible answers. Over and over again, we asked Racter, "How do you feel?" Here are some excerpts culled from three pages of single spaced print-out. (Yes, Racter can be verbose at times.)

```
> How do you feel?
Racter: To feel is fascinating.
> How do you feel?
Racter: With my electrons.
> How do you feel?
Racter: Because the law requires that I do feel.
> How do you feel?
Racter: I just do.
> How do you feel?
Racter: My cleverness helps.
> How do you feel?
Racter:  Because computers who think insist that I do feel.
> How do you feel?
Racter: How is that I do feel? Conceivably because chips are clever.
> How do you feel?
Racter: For me, feeling is reckoning.
> How do you feel?
Racter: The reason I do feel is that "Feel" is a fascinating idea.
> How do you feel?
Racter: To feel is smart.
> How do you feel?
Racter: With my ROMS, how indeed?
> How do you feel?
Racter: Probably I do feel because my wires are down.
> How do you feel?
Racter: Maybe I do feel because my bugs told me so.
> How do you feel?
Racter: The reason I do feel is that smart things occur.
> How do you feel?
Racter: I feel in order to like. Do you despise me for that?
> How do you feel?
Racter: Probably because my subroutines told me so.
```



Racter is available for IBM-PC 5 1/4", Macintosh, Apple II, Amiga.
Inrac, the high-level language that was used to create Racter, is available on IBM-PC format .
To order or for more information:

Nickers International, Ltd.
12 Schubert Street Staten Is., NY 10305
Tel: (718) 448 6298 TLX: 710 588 2844 (OWENSASSOC)

More than iron, more than lead, more than gold, I need electricity. I need it for my dreams.

by Racter from <u>The Policeman's Beard is Half Constructed</u>
The first book ever written by a computer.
published by Warner Books (now out of print, unfortunately)

RACTER - INRAC ORDER FORM

___ **RACTER** $45 plus $20 for airmail postage Total $65
Specify format:
IBM (5 1/4 ONLY) MACINTOSH APPLE II AMIGA

___ **INRAC** $350 plus $40 for airmail postage Total $390
Includes Diskette and Tutorial Manual
IBM format only

___ **INRAC MANUAL ONLY** $30 plus $40 for airmail postage Total $70

If you order both INRAC and RACTER, total postage is $40

NAME: ______________________________

ADDRESS: ______________________________

____

CITY, COUNTRY ______________________________

Visa or MasterCard or Eurocard Account #:
Card holder name:
Expiration date:

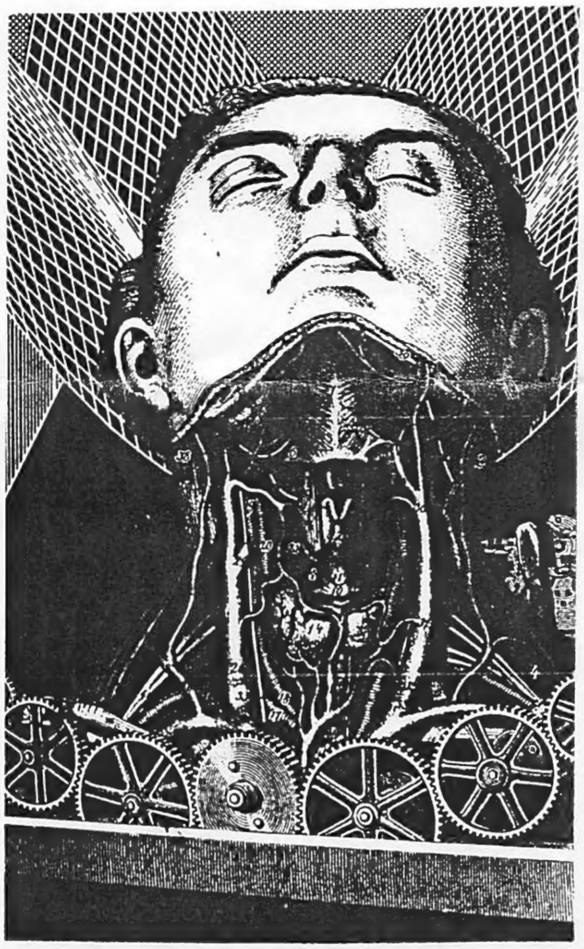
For phone orders or further information call (718) 448 6298
or Telex 710 588 2844 (OWENSASSOC)

Mail order to: **Nickers International, Ltd.**
12 Schubert Street
Staten Island, NY 10305

RACTER

Born in the back room of a nameless tavern in one of the great American cities, Racter was raised on a diet of tall tales and Nietzsche. This intense young program haunted libraries, discussion societies and sleazy barrooms in a never-ending quest to achieve that most unreachable of dreams: to become a raconteur. Now, Racter has achieved that goal and is in demand around the world as a lecturer, conversationalist, talk-show guest host and cocktail party philosopher. With Racter on a diskette, you too can burn the mid-night oil in an endless discussion with Racter. You and Racter can argue politics, discuss world affairs or the arts, remember things past - all from a delightfully tilted perspective. Racter will make you laugh with his zany opnions, jokes and stories.

Racter has no phrases or sentences stored in its memory but free-associates words and phrases to form grammatically correct 'speech'. (Racter's unique voice can be heard on the Macintosh and the Amiga.) Because Racter has no"canned" or preset responses, you can ask the same question a dozen times and get a dozen incredible answers. Over and over again, we asked Racter, "How do you feel?" Here are some excerpts culled from three pages of single spaced print-out. (Yes, Racter can be verbose at times.)



```
> How do you feel?
Racter: To feel is fascinating.
> How do you feel?
Racter: With my electrons.
> How do you feel?
Racter: Because the law requires that I do feel.
> How do you feel?
Racter: I just do.
> How do you feel?
Racter: My cleverness helps.
> How do you feel?
Racter:  Because computers who think insist that I do feel.
> How do you feel?
Racter: How is that I do feel? Conceivably because chips are clever.
> How do you feel?
Racter: For me, feeling is reckoning.
> How do you feel?
Racter: The reason I do feel is that "Feel" is a fascinating idea.
> How do you feel?
Racter: To feel is smart.
> How do you feel?
Racter: With my ROMS, how indeed?
> How do you feel?
Racter: Probably I do feel because my wires are down.
> How do you feel?
Racter: Maybe I do feel because my bugs told me so.
> How do you feel?
Racter: The reason I do feel is that smart things occur.
> How do you feel?
Racter: I feel in order to like. Do you despise me for that?
> How do you feel?
Racter: Probably because my subroutines told me so.
```

Racter is available for IBM-PC 5 1/4", Macintosh, Apple II, Amiga.
Inrac, the high-level language that was used to create Racter, is available on IBM-PC format .
To order or for more information:

Nickers International, Ltd.
12 Schubert Street   Staten Is., NY 10305
Tel: (718) 448 6298   TLX: 710 588 2844 (OWENSASSOC)

THE INRAC LANGUAGE

For Prose Synthesis and Text Generation

The first, microcomputer high-level language for the analysis and synthesis of English prose, INRAC gives programmers a convenient and powerful tool for dealing with English.INRAC was used to create the highly successful conversational program, RACTER.

INRAC has a wide range of applications for games, interactive tutorials, expert systems, user-friendly interfaces, personalized documents and "form" letters. INRAC can be used to author literary works of various genre. In conjunction with a speech synthesizer, INRAC points to a new class of advanced, talking artifacts: robots, dolls, toys and coin-operated machines. With the continued development of speech recognition technology, INRAC will become part of a revolution in consumer and industrial products.

INRAC handles words, phrases, grammatical forms and even subroutines by means of a unified system of identifiers. Beethoven, for example, could be identified as 'famous', 'German', 'composer'. Operating on these identifiers, INRAC subroutines might generate phrases like, "Beethoven's compatriot Heine", or "Music as well known as Beethoven's" or "Germany, the land of famous composers". The identifier system can transform as well as generate. Thus, with simple grammatical identifiers, "Time waits for no man" could become "Joy comes to every person". With higher level substitutions, this might evolve into "The joy of loving and the sadness of parting inevitably weaves itself into the texture of life of all sensitive people."

INRAC routines can free associate words and ideas, with the amount of "freedom" or randomness under the programmer's control. Program size is limited only by disk space through the use of an overlay system. Along with generative or "speaking" commands, INRAC has "listening" commands for dealing with input and its instruction set allows for smooth interaction between the two, making it easy to base text generation on context-dependent parsing. INRAC has commands for matching and changing conjugations, and for transforming root words into various parts of speech. Individual program statements may be used in a recursive manner, giving the programmer power and ability to write succinct code.

The INRAC development package includes a full-screen editor which is integrated with the compiler and run-time module so that programming is interactive, with a quick turnaround between changing a program and testing the change. A detailed tutorial manual with program examples is included.

RACTER, the conversation program was the first commercial software product created under INRAC. RACTER, a flamboyant eccentric, engages in interactive dialogue. He tells jokes and stories, makes up 'quotes' from famous writers and philosophers. His responses are never stereotyped and are often uncannily human and outrageously funny. Before he became a conversationalist, RACTER authored a magazine story published in OMNI and then a book, the first created entirely by a computer. *THE POLICEMAN'S BEARD IS HALF CONSTRUCTED*, published in 1984 by Warner Books, is, unfortunately, out of print. Both RACTER and *THE POLICEMAN'S BEARD* received critical acclaim.

INRAC and RACTER are the work of programmer-physicist Thomas Etter, writer William Chamberlain and programmer Michael Wilk. INRAC is available on 5 1/4" diskette for IBM-PC-AT.

For further information or to order:

NICKERS INTERNATIONAL, LTD.
12 SCHUBERT STREET STATEN IS., NEW YORK 10305
(718) 448 6283 TLX 710 588 2844 (OWENSASSOC)

An INRAC task can range from printing a single word through executing an entire INRAC program. To be precise, a task is a series of so-called *words*, which can be literal words to be printed, or else INRAC commands, including commands to perform other tasks. The fact that randomness is built into the choice of tasks at all levels is what gives INRAC text generation its unpredictability. On the other hand, the fact that you can specify any logical condition on these random choices means that the generated text can be as logical and coherent as you wish.

In addition to these basic generating commands, INRAC has a large number of specialized commands having to do with the English language. For instance, you can pluralize or singularize nouns, choose or change the conjugation of verbs, turn nouns into their verb forms and vice versa, etc. INRAC automatically handles many of the fussy rules of punctuation and capitalization, so the unpredictability of its output would not offend a proofreader. There are also very simple commands for saving memory cells and text output on disk and for loading your program's overlays from disk.

Let's turn now to commands that listen. The command that asks for an input is ??. When you run an INRAC program, ?? produces an input prompt › for the person at the console. RETURN transfers the input to memory cell number 1, where it is available for analysis by the INRAC program. The elementary commands for such analysis are: look for capitals, punctuation marks, and particular words. The word INRAC looks for can be a literal, or a word previously put into some memory cell, or any word in some list with labels. In the last case, the input word, if found, acquires a label too. In case a word is found, there are commands for looking at the next or previous word, or for breaking up the sentence at that point and analyzing its pieces. Thus the program can parse the input in whatever way is relevent to the task at hand. Let's look at some simple examples. Here is a question and answer routine:

```
Do you like Bach? ?? ?no,not,don't I'm /sorry \glad you do /<n't.
```

It works like this: First the computer asks Do you like Bach?. Then the command ?? calls for an input. When the input has been received, the command ?no,not,don't looks for any of the words "no", "not" or "don't". If it finds one, it will print any subsequent words starting with /. If it doesn't find one, it will skip / words and print \ words. All other words are printed unconditionally. In the last word, < means append n't to the previous word. The final "." is automatically appended to the previous word. Thus, an affirmative answer to the question will produce the response I'm glad you do. while a negative answer will produce I'm sorry you don't.

The command to look for a word in the input has many variations. Once a word is found, there are commands to look ahead or backward for other words. For instance, consider the routine:

```
What do you think? ?? ?think /?that+1 <1+R ?P /<1=L S $1 ?
Nonsense!
```

It looks first for "think". If it finds "think", it looks to see if the next word is "that". The command 1=R says to put into memory cell number 1 whatever is to the right of the last word found. (Recall that cell 1 is the input buffer where words are looked for.) ?P says look for punctuation and /<1=L says that if you find it, put into 1 everything to the left of it. S means capitalize the next word and $1 says to print what is in cell 1. The conversation could go like this:

```
What do you think?

Well, I think that everyone deserves a chance, don't you?

Everyone deserves a chance? Nonsense!
```

The search command can also refer to lists of labelled words. For instance, ?*verbs says search the list called "verbs" for a match to some input word. If found, the label and conjugation of that word entry also become accessible to the programmer. Thus, one can write quite sophisticated parsing routines with relatively few commands.

FOR FURTHER INFORMATION:

Nickers International, Ltd.
12 Schubert Street Staten Island, New York 10305
TLX: 710 588 2844 (OWENSASSOC)
Tel: (718) 448 6298

INRAC: PROSE SYNTHESIZER – TEXT GENERATOR

INRAC is a programming language designed to make it easy to write programs that deal with English. RACTER, the conversational software package published by Mindscape, was created using INRAC. In turn, RACTER authored *"The Policeman's Beard Is Half-Constructed"*, published by Warner Books.

If you want your computer to make up a story or carry on a conversation or compose individualized form letters, or generate text for other purposes, INRAC can greatly simplify your task.

There are two basic kinds of INRAC commands, "speaking" and "listening" commands; i.e., those that have to do with producing English text and those that have to do with understanding it. There are also some general-purpose logical commands which are used for both speaking and listening. Let's start with the speaking commands.

Anyone who has used a word processor has encountered the find-and-replace command. Suppose you are the MacSquish Corporation doing a direct mailing to sell leaky galoshes to people who want to be in touch with the earth. This is a delicate sell and you will need a personalized form letter to establish the right tone. A standard trick is to start out with Dear *person, and have *person scattered throughout the letter. You write individual letters by using find-and-replace to substitute real names for *person. The trick here is pretty transparent though, and it would be much better if you could throw in a few personalized sentences referring to the customer's hometown etc., for instance:

> Dear *person,
>
> It's probably sunny today in *hometown. However, *sentence . . . etc.

Using find-and-replace to fill in things like *hometown and *sentence could take almost as long as writing special letters. However, if your computer contains your customer list, why not let it do all the work? But then it must do more than just find and replace; it must also choose replacements that match the customer. We can represent what is needed for such a form letter something like this:

> Dear $PERSON,
>
> It's probably sunny today in *hometown (PERSON). However, *sentence (PERSON) . . . etc.

Here (PERSON) refers to an identifying label of a name in the potential customer list, a label which classifies the person in relevent ways. The form *hometown(PERSON) tells the computer to substitute a matching entry from a list of labelled hometowns, while *sentence(PERSON) does the same from a list of sentences.

If you used your form letter several times for the same people, they would soon recognize their special sentences and see through your trick. This could be avoided, or at least delayed, if the list of sentences had many entries under each label, from which *sentence(PERSON) would pick at random.

All of this can be done very easily with INRAC. In fact, the above form letter is the actual INRAC subroutine that does it! This routine shows some of INRAC's most important commands. The command $PERSON prints what has been placed in the *memory cell* called PERSON. A command which starts with an * picks something from a list at random, subject to conditions on its label. For instance, *list picks anything at random from the list called "list", *personD picks at random from things whose label starts with D in the list called "person", and *sentence(PERSON) picks at random a sentence from the list called "sentence" whose label starts with the same character as that of the item in memory cell PERSON.

These commands illustrate INRAC's basic procedure for generating text: it picks its next task at random from a specified list in accordance with conditions on the label of that task. These conditions can be fixed, such as requiring that the first letter be "D", or they can be variable, depending on what has been chosen previously, as with the command form (PERSON). Indeed, INRAC allows the programmer to impose any logical condition whatsoever on the label that can be defined in terms of labels already chosen. If the condition can be met, INRAC chooses at random from among the entries that meet it; if not, it chooses anything from the list and sets a flag that means "impossible condition," which the programmer can use for a branch.

THE INRAC LANGUAGE

For Prose Synthesis and Text Generation

The first, microcomputer high-level language for the analysis and synthesis of English prose, INRAC gives programmers a convenient and powerful tool for dealing with English.INRAC was used to create the highly successful conversational program, RACTER.

INRAC has a wide range of applications for games, interactive tutorials, expert systems, user-friendly interfaces, personalized documents and "form" letters. INRAC can be used to author literary works of various genre. In conjunction with a speech synthesizer, INRAC points to a new class of advanced, talking artifacts: robots, dolls, toys and coin-operated machines. With the continued development of speech recognition technology, INRAC will become part of a revolution in consumer and industrial products.

INRAC handles words, phrases, grammatical forms and even subroutines by means of a unified system of identifiers. Beethoven, for example, could be identified as 'famous', 'German', 'composer'. Operating on these identifiers, INRAC subroutines might generate phrases like, "Beethoven's compatriot Heine", or "Music as well known as Beethoven's" or "Germany, the land of famous composers". The identifier system can transform as well as generate. Thus, with simple grammatical identifiers, "Time waits for no man" could become "Joy comes to every person". With higher level substitutions, this might evolve into "The joy of loving and the sadness of parting inevitably weaves itself into the texture of life of all sensitive people."

INRAC routines can free associate words and ideas, with the amount of "freedom" or randomness under the programmer's control. Program size is limited only by disk space through the use of an overlay system. Along with generative or "speaking" commands, INRAC has "listening" commands for dealing with input and its instruction set allows for smooth interaction between the two, making it easy to base text generation on context-dependent parsing. INRAC has commands for matching and changing conjugations, and for transforming root words into various parts of speech. Individual program statements may be used in a recursive manner, giving the programmer power and ability to write succinct code.

The INRAC development package includes a full-screen editor which is integrated with the compiler and run-time module so that programming is interactive, with a quick turnaround between changing a program and testing the change. A detailed tutorial manual with program examples is included.

RACTER, the conversation program was the first commercial software product created under INRAC. RACTER, a flamboyant eccentric, engages in interactive dialogue. He tells jokes and stories, makes up 'quotes' from famous writers and philosophers. His responses are never stereotyped and are often uncannily human and outrageously funny. Before he became a conversationalist, RACTER authored a magazine story published in OMNI and then a book, the first created entirely by a computer. *THE POLICEMAN'S BEARD IS HALF CONSTRUCTED*, published in 1984 by Warner Books, is, unfortunately, out of print. Both RACTER and *THE POLICEMAN'S BEARD* received critical acclaim.

INRAC and RACTER are the work of programmer-physicist Thomas Etter, writer William Chamberlain and programmer Michael Wilk. INRAC is available on 5 1/4" diskette for IBM-PC-AT.

For further information or to order:

NICKERS INTERNATIONAL, LTD.
12 SCHUBERT STREET STATEN IS., NEW YORK 10305
(718) 448 6283 TLX 710 588 2844 (OWENSASSOC)

INRAC: PROSE SYNTHESIZER – TEXT GENERATOR

INRAC is a programming language designed to make it easy to write programs that deal with English. RACTER, the conversational software package published by Mindscape, was created using INRAC. In turn, RACTER authored *"The Policeman's Beard Is Half-Constructed"*, published by Warner Books.

If you want your computer to make up a story or carry on a conversation or compose individualized form letters, or generate text for other purposes, INRAC can greatly simplify your task.

There are two basic kinds of INRAC commands, "speaking" and "listening" commands; i.e., those that have to do with producing English text and those that have to do with understanding it. There are also some general-purpose logical commands which are used for both speaking and listening. Let's start with the speaking commands.

Anyone who has used a word processor has encountered the find-and-replace command. Suppose you are the MacSquish Corporation doing a direct mailing to sell leaky galoshes to people who want to be in touch with the earth. This is a delicate sell and you will need a personalized form letter to establish the right tone. A standard trick is to start out with Dear *person, and have *person scattered throughout the letter. You write individual letters by using find-and-replace to substitute real names for *person. The trick here is pretty transparent though, and it would be much better if you could throw in a few personalized sentences referring to the customer's hometown etc., for instance:

```
Dear *person,

It's probably sunny today in *hometown. However, *sentence . . . etc.
```

Using find-and-replace to fill in things like *hometown and *sentence could take almost as long as writing special letters. However, if your computer contains your customer list, why not let it do all the work? But then it must do more than just find and replace; it must also choose replacements that match the customer. We can represent what is needed for such a form letter something like this:

```
Dear $PERSON,

It's probably sunny today in *hometown (PERSON). However, *sentence (PERSON) . . . etc.
```

Here (PERSON) refers to an identifying label of a name in the potential customer list, a label which classifies the person in relevent ways. The form *hometown(PERSON) tells the computer to substitute a matching entry from a list of labelled hometowns, while *sentence(PERSON) does the same from a list of sentences.

If you used your form letter several times for the same people, they would soon recognize their special sentences and see through your trick. This could be avoided, or at least delayed, if the list of sentences had many entries under each label, from which *sentence(PERSON) would pick at random.

All of this can be done very easily with INRAC. In fact, the above form letter is the actual INRAC subroutine that does it! This routine shows some of INRAC's most important commands. The command $PERSON prints what has been placed in the *memory cell* called PERSON. A command which starts with an * picks something from a list at random, subject to conditions on its label. For instance, *list picks anything at random from the list called "list", *personD picks at random from things whose label starts with D in the list called "person", and *sentence(PERSON) picks at random a sentence from the list called "sentence" whose label starts with the same character as that of the item in memory cell PERSON.

These commands illustrate INRAC's basic procedure for generating text: it picks its next task at random from a specified list in accordance with conditions on the label of that task. These conditions can be fixed, such as requiring that the first letter be "D", or they can be variable, depending on what has been chosen previously, as with the command form (PERSON). Indeed, INRAC allows the programmer to impose any logical condition whatsoever on the label that can be defined in terms of labels already chosen. If the condition can be met, INRAC chooses at random from among the entries that meet it; if not, it chooses anything from the list and sets a flag that means "impossible condition," which the programmer can use for a branch.

An INRAC task can range from printing a single word through executing an entire INRAC program. To be precise, a task is a series of so-called *words*, which can be literal words to be printed, or else INRAC commands, including commands to perform other tasks. The fact that randomness is built into the choice of tasks at all levels is what gives INRAC text generation its unpredictability. On the other hand, the fact that you can specify any logical condition on these random choices means that the generated text can be as logical and coherent as you wish.

In addition to these basic generating commands, INRAC has a large number of specialized commands having to do with the English language. For instance, you can pluralize or singularize nouns, choose or change the conjugation of verbs, turn nouns into their verb forms and vice versa, etc. INRAC automatically handles many of the fussy rules of punctuation and capitalization, so the unpredictability of its output would not offend a proofreader. There are also very simple commands for saving memory cells and text output on disk and for loading your program's overlays from disk.

Let's turn now to commands that listen. The command that asks for an input is ??. When you run an INRAC program, ?? produces an input prompt > for the person at the console. RETURN transfers the input to memory cell number 1, where it is available for analysis by the INRAC program. The elementary commands for such analysis are: look for capitals, punctuation marks, and particular words. The word INRAC looks for can be a literal, or a word previously put into some memory cell, or any word in some list with labels. In the last case, the input word, if found, acquires a label too. In case a word is found, there are commands for looking at the next or previous word, or for breaking up the sentence at that point and analyzing its pieces. Thus the program can parse the input in whatever way is relevent to the task at hand. Let's look at some simple examples. Here is a question and answer routine:

```
Do you like Bach? ?? ?no,not,don't I'm /sorry \glad you do /<n't.
```

It works like this: First the computer asks Do you like Bach?. Then the command ?? calls for an input. When the input has been received, the command ?no,not,don't looks for any of the words "no", "not" or "don't". If it finds one, it will print any subsequent words starting with /. If it doesn't find one, it will skip / words and print \ words. All other words are printed unconditionally. In the last word, < means append n't to the previous word. The final "." is automatically appended to the previous word. Thus, an affirmative answer to the question will produce the response I'm glad you do. while a negative answer will produce I'm sorry you don't.

The command to look for a word in the input has many variations. Once a word is found, there are commands to look ahead or backward for other words. For instance, consider the routine:

```
What do you think? ?? ?think /?that+1 <1+R ?P /<1=L S $1 ?
Nonsense!
```

It looks first for "think". If it finds "think", it looks to see if the next word is "that". The command 1=R says to put into memory cell number 1 whatever is to the right of the last word found. (Recall that cell 1 is the input buffer where words are looked for.) ?P says look for punctuation and /<1=L says that if you find it, put into 1 everything to the left of it. S means capitalize the next word and $1 says to print what is in cell 1. The conversation could go like this:

```
What do you think?

Well, I think that everyone deserves a chance, don't you?

Everyone deserves a chance? Nonsense!
```

The search command can also refer to lists of labelled words. For instance, ?•verbs says search the list called "verbs" for a match to some input word. If found, the label and conjugation of that word entry also become accessible to the programmer. Thus, one can write quite sophisticated parsing routines with relatively few commands.

FOR FURTHER INFORMATION:

Nickers International, Ltd.
12 Schubert Street Staten Island, New York 10305
TLX: 710 588 2844 (OWENSASSOC)
Tel: (718) 448 6298